Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по рубежному контролю №2 по курсу БКИТ

Выполнил:
студент группы ИУ5-31Б
Бутрим Андрей Александрович

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Подпись и дата:

Подпись и дата:

Москва, 2021 г.

Рубежный контроль№2
Задание

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD – фреймворка (3 теста).

Текст программы
main.py

```
from operator import itemgetter


class Dri:
    """Водитель"""
    """Переменная hours - кол-во часов работы в неделю"""
    def __init__(self, id, fio, hours, park_id):
        self.id = id
        self.fio = fio
        self.hours = hours
        self.park_id = park_id


class Park:
    """Автопарк"""

    def __init__(self, id, name):
        self.id = id
        self.name = name


class DriPark:
    def __init__(self, park_id, dri_id):
        self.park_id = park_id
        self.dri_id = dri_id


# Автопарки
parks = [
    Park(1, 'Яндекс такси'),
    Park(2, 'Ситимобил'),
    Park(3, 'Gett такси'),

    Park(11, 'Uber такси'),
    Park(22, 'Автопарк-Гарант'),
    Park(33, 'Калита')
]

# Водители
drivers = [
    Dri(1, 'Рыжов', 50, 1),
    Dri(2, 'Спиряков', 25, 2),
    Dri(3, 'Осипов', 15, 3),
    Dri(4, 'Медведев', 35, 3),
    Dri(5, 'Гончаров', 55, 3),
]

drivers_parks = [
    DriPark(1, 1),
    DriPark(2, 2),
    DriPark(3, 3),
    DriPark(3, 4),
    DriPark(3, 5),
```

```
    DriPark(11, 1),
    DriPark(22, 2),
    DriPark(33, 3),
    DriPark(33, 4),
    DriPark(33, 5),
]


def sorting_by_name(table):
    return sorted(table, key=itemgetter(2))


def sorting_by_sum_hours(table, parks):
    result_unsorted = []
    # Перебираем все парки
    for p in parks:
        # Список водителей парка
        p_drivers = list(filter(lambda i: i[2] == p.name, table))
        # Если отдел не пустой
        if len(p_drivers) > 0:
            # Зарплаты водителей автопарка
            p_hours = [hours for _, hours, _ in p_drivers]
            # Суммарная зарплата водителей автопарка
            p_hours_sum = sum(p_hours)
            result_unsorted.append((p.name, p_hours_sum))
    return sorted(result_unsorted, key=itemgetter(1), reverse=True)


def output_drivers_of_parks_with_TAXI(table, parks):
    result = {}
    # Перебираем все автопарки
    for p in parks:
        if 'такси' in p.name:
            # Список водителей автопарка
            p_drivers = list(filter(lambda i: i[2] == p.name, table))
            # Только ФИО сотрудников
            p_drivers_names = [x for x, _, _ in p_drivers]
            # Добавляем результат в словарь
            # ключ - автопарк, значение - список фамилий
            result[p.name] = p_drivers_names
    return result

def main():
    """Основная функция"""

    # Соединение данных один-ко-многим
    one_to_many = [(d.fio, d.hours, p.name)
                   for p in parks
                   for d in drivers
                   if d.park_id == p.id]

    # Соединение данных многие-ко-многим
    many_to_many_temp = [(p.name, dp.park_id, dp.dri_id)
                         for p in parks
                         for dp in drivers_parks
                         if p.id == dp.park_id]

    many_to_many = [(d.fio, d.hours, park_name)
                    for park_name, park_id, dri_id in many_to_many_temp
                    for d in drivers if d.id == dri_id]

    print('Задание А1')
    print("#Список связанных водителей и автопарков, отсортированный по автопаркам")
    print(sorting_by_name(one_to_many))

    print('\nЗадание А2')
    print("#Список автопарков и суммарным количеством часов работы их водителей,")
```

```
    print("отсортированный по кол-ву часов")
    print(sorting_by_sum_hours(one_to_many, parks))

    print('\nЗадание А3')
    print("#Список всех автопарков, у которых в названии есть 'такси' и список всех их
водителей")
    print(output_drivers_of_parks_with_TAXI(many_to_many, parks))


if __name__ == '__main__':
    main()
```

tests.py

```
from main import Dri, Park, DriPark, sorting_by_name, sorting_by_sum_hours,
output_drivers_of_parks_with_TAXI
import unittest


class Tests(unittest.TestCase):
    def setUp(self):
        # Автопарки
        self.parks = [
            Park(1, 'Яндекс такси'),
            Park(2, 'Ситимобил'),
            Park(3, 'Gett такси'),

            Park(11, 'Uber такси'),
            Park(22, 'Автопарк-Гарант'),
            Park(33, 'Калита')
        ]

        # Водители
        self.drivers = [
            Dri(1, 'Рыжов', 50, 1),
            Dri(2, 'Спиряков', 25, 2),
            Dri(3, 'Осипов', 15, 3),
            Dri(4, 'Медведев', 35, 3),
            Dri(5, 'Гончаров', 55, 3),
        ]

        self.drivers_parks = [
            DriPark(1, 1),
            DriPark(2, 2),
            DriPark(3, 3),
            DriPark(3, 4),
            DriPark(3, 5),

            DriPark(11, 1),
            DriPark(22, 2),
            DriPark(33, 3),
            DriPark(33, 4),
            DriPark(33, 5),
        ]

        # Соединение данных один-ко-многим
        self.one_to_many = [(d.fio, d.hours, p.name)
                            for p in self.parks
                            for d in self.drivers
                            if d.park_id == p.id]

        # Соединение данных многие-ко-многим
        self.many_to_many_temp = [(p.name, dp.park_id, dp.dri_id)
                                  for p in self.parks
                                  for dp in self.drivers_parks
                                  if p.id == dp.park_id]
```

```
        self.many_to_many = [(d.fio, d.hours, park_name)
                             for park_name, park_id, dri_id in self.many_to_many_temp
                             for d in self.drivers if d.id == dri_id]

    def test_sorting_by_name(self):
        result = sorting_by_name(self.one_to_many)
        desired_result = [('Осипов', 15, 'Gett такси'), ('Медведев', 35, 'Gett такси'),
('Гончаров', 55, 'Gett такси'),
                          ('Спиряков', 25, 'Ситимобил'), ('Рыжов', 50, 'Яндекс такси')]
        self.assertEqual(result, desired_result)

    def test_sorting_by_sum(self):
        result = sorting_by_sum_hours(self.one_to_many, self.parks)
        desired_result = [('Gett такси', 105), ('Яндекс такси', 50), ('Ситимобил', 25)]
        self.assertEqual(result, desired_result)

    def test_output_PAPKA2(self):
        result = output_drivers_of_parks_with_TAXI(self.many_to_many, self.parks)
        desired_result = {'Яндекс такси': ['Рыжов'], 'Gett такси': ['Осипов',
'Медведев', 'Гончаров'],
                          'Uber такси': ['Рыжов']}
        self.assertEqual(result, desired_result)
```

Результат выполнения программы:

```
Задание A1
#Список связанных водителей и автопарков, отсортированный по автопаркам
[('Осипов', 15, 'Gett такси'), ('Медведев', 35, 'Gett такси'), ('Гончаров', 55, 'Gett такси'), ('Спиряков', 25, 'Ситимобил'), ('Рыжов', 50, 'Яндекс

Задание A2
#Список автопарков и суммарным количеством часов работы их водителей,
отсортированный по кол-ву часов
[('Gett такси', 105), ('Яндекс такси', 50), ('Ситимобил', 25)]

Задание A3
#Список всех автопарков, у которых в названии есть 'такси' и список всех их водителей
{'Яндекс такси': ['Рыжов'], 'Gett такси': ['Осипов', 'Медведев', 'Гончаров'], 'Uber такси': ['Рыжов']}
```